Guía de Usuario de Arduino

Rafael Enríquez Herrador

13 de noviembre de 2009

I.T.I. Sistemas

Universidad de Córdoba

i52enher@uco.es



Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

PREFACIO

Esta guía de usuario intenta ser una forma de acercarse al diseño y desarrollo de proyectos basados en Arduino para aquellas personas que nunca han trabajado con él pero que poseen un buen nivel en programación y electrónica. Por esta razón y para hacerlo fácil, se ha excluido mucha información existente en Internet y otros manuales para centrarse en los aspectos más básicos de las características y la programación de Arduino.

Otro de los objetivos de esta guía es organizar un poco la gran cantidad de información sobre este tema existente en la red. Para ello casi toda la información se ha obtenido a través de la fuente `http://www.arduino.cc` o de manuales basados en ella pero algo más estructurados. En general, el texto es una traducción libre al español del documento original *«Arduino Programming Notebook»* escrito y compilado por «Brian W. Evans».

Por último, la guía está pensada para aquellas personas que no han usado Arduino pero les gustaría iniciarse en este campo, por lo que si eres un usuario avanzado de esta plataforma no te aportará nada nuevo (sólo te servirá para repasar conceptos básicos).

Espero que les sea de utilidad.

Capítulo 2

INTRODUCCIÓN

2.1. ¿Qué es ARDUINO?

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede «sentir» el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el «Arduino Programming Language» (basado en Wiring[1]) y el «Arduino Development Environment» (basado en Processing[2]). Los proyectos de Arduino pueden ser autonomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con Flash, Processing, MaxMSP, etc.).

Las placas se pueden ensamblar a mano[3] o encargarlas preensambladas[4]; el software se puede descargar[5] gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.

Arduino recibió una mención honorífica en la sección Digital Communities del Ars Electronica Prix en 2006.

2.2. ¿Por qué ARDUINO?

Hay muchos otros microcontroladores y plataformas microcontroladoras disponibles para computación física. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, y muchas otras ofertas de funcionalidad similar. Todas estas herramientas toman los desordenados detalles de la programación de microcontrolador y la encierran en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes y aficionados interesados sobre otros sistemas:

[1] Más información en `http://wiring.org.co`
[2] Más información en `http://www.processing.org`
[3] Más información en `http://www.arduino.cc/en/Main/USBAssembly`
[4] Más información en `http://www.arduino.cc/en/Main/Buy`
[5] Más información en `http://www.arduino.cc/en/Main/Software`

- Barato: Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras. La versión menos cara del modulo Arduino puede ser ensamblada a mano, e incluso los módulos de Arduino preensamblados cuestan menos de 50$.

- Multiplataforma: El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas microcontroladores están limitados a Windows.

- Entorno de programación simple y claro: El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios avanzados puedan aprovecharlo también. Para profesores, está convenientemente basado en el entorno de programación Processing, de manera que estudiantes aprendiendo a programar en ese entorno estarán familiarizados con el aspecto y la imagen de Arduino.

- Código abierto y software extensible: El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados. El lenguaje puede ser expandido mediante librerias C++, y la gente que quiera entender los detalles técnicos pueden hacer el salto desde Arduino a la programación en lenguaje AVR C en el cual está basado. De forma similar, puedes añadir código AVR-C directamente en tus programas Arduino si quieres.

- Código abierto y hardware extensible: El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo. Incluso usuarios relativamente inexpertos pueden construir la versión de la placa del módulo para entender como funciona y ahorrar dinero.

Capítulo 3

HARDWARE

Hay múltiples versiones de la placa Arduino. La mayoría usan el ATmega168 de Atmel, mientras que las placas más antiguas usan el ATmega8.

Nota: Los diseños de referencia para Arduino se distribuyen bajo licencia Creative Commons Attribution-ShareAlike 2.5.

3.1. Placas E/S

- **Diecimila:** Esta es la placa Arduino más popular. Se conecta al ordenador con un cable estándar USB y contiene todo lo que necesitas para programar y usar la placa. Puede ser ampliada con variedad de dispositivos: placas hijas con características específicas.

- **Nano:** Una placa compacta diseñada para uso como tabla de pruebas, el Nano se conecta al ordenador usando un cable USB Mini-B.

- **Bluetooth:** El Arduino BT contiene un modulo bluetooth que permite comunicación y programación sin cables. Es compatible con los dispositivos Arduino.

- **LilyPad:** Diseñada para «aplicaciones listas para llevar», esta placa puede ser conectada en fábrica, y un estilo sublime.

- **Mini:** Esta es la placa más pequeña de Arduino. Trabaja bien en tabla de pruebas o para aplicaciones en las que prima el espacio. Se conecta al ordenador usando el cable Mini USB.

- **Serial:** Es una placa básica que usa RS232 como un interfaz con el ordenador para programación y comunicación. Esta placa es fácil de ensamblar incluso como ejercicio de aprendizaje.

- **Serial Single Sided:** Esta placa está diseñada para ser grabada y ensamblada a mano. Es ligeramente más grande que la Diecimila, pero aun compatible con los dispositivos.

3.2. Arduino Diecimila

3.2.1. Visión General

El Arduino Diecimila es una placa microcontroladora basada en el ATmega168. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden ser usados como salidas PWM), 6 entradas analógicas, un oscilador de cuarzo a 16MHz, una conexión USB, un conector para alimentación, una cabecera ICSP, y un botón de reset. Contiene todo lo necesario para soportar el microcontrolador; simplemente conéctalo a un ordenador con un cable USB o enchúfalo con un adaptador AC/DC o batería para comenzar.

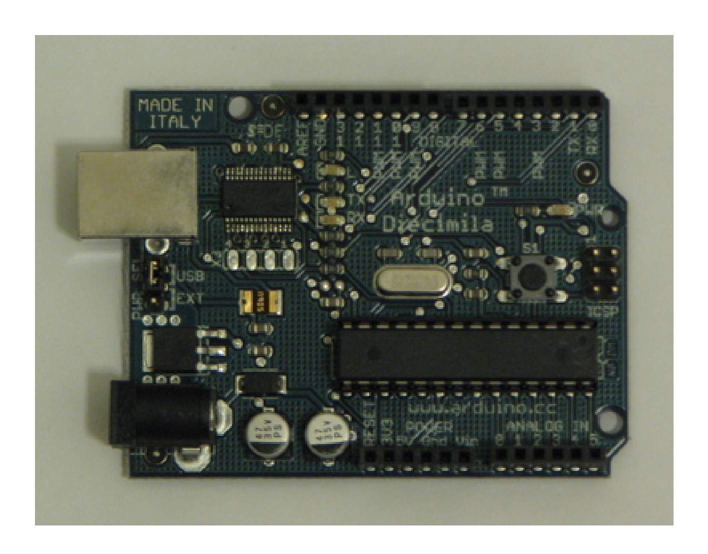


Figura 3.1: Placa Arduino Diecimila (USB)

«Diecimila» quiere decir 10000 en italiano y fue llamado así para resaltar el hecho de que más de 10000 placas Arduino han sido fabricadas. El Diecimila es el último en la serie de placas USB Arduino.

3.2.2. Resumen

Característica	**Descripción**
Microcontrolador	ATmega168
Voltaje de operación	5 V
Tensión de entrada (recomendada)	7 - 12 V
Tensión de entrada (límite)	6 - 20 V
Pines digitales de E/S	14 (de los cuales 6 proveen salidas PWM)
Pines de entrada analógicos	6
Corriente DC por pin E/S	40 mA
Corriente DC para pin 3.3 V	50 mA
Memoria Flash	16 KB (de los cuales 2 KB usados para bootloader)
SRAM	1 KB
EEPROM	512 bytes
Frecuencia de reloj	16 MHz

Cuadro 3.1: Características técnicas de Arduino Diecimila

3.2.3. Alimentación

El Arduino Diecimila puede ser alimentado a través de la conexión USB o con un suministro de energía externo. La fuente de energía se selecciona mediante el jumper PWR_SEL: para alimentar a la placa desde la conexión USB, colocarlo en los dos pines más cercanos al conector USB, para un suministro de energía externo, en los dos pines más cercanos al conector de alimentación externa.

La alimentación externa (no USB) puede venir o desde un adaptador AC-a-DC (wall-wart) o desde una batería. El adaptador puede ser conectado mediante un enchufe centro-positivo en el conector de alimentación de la placa. Los cables de la batería pueden insertarse en las cabeceras de los pines Gnd y Vin del conector POWER. Un regulador de bajo abandono proporciona eficiencia energética mejorada.

La placa puede operar con un suministro externo de 6 a 20 voltios. Si es suministrada con menos de 7 V, sin embargo, el pin de 5 V puede suministrar menos de cinco voltios y la placa podría ser inestable. Si usa más de 12 V, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- **VIN**. La entrada de tensión a la placa Arduino cuando está usando una fuente de alimentación externa (al contrario de los 5 voltios de la conexión USB u otra fuente de alimentación regulada). Puedes suministrar tensión a través de este pin, o, si suministra tensión a través del conector de alimentación, acceder a él a través de este pin.

- **5V.** El suministro regulado de energía usado para alimentar al microcontrolador y otros componentes de la placa. Este puede venir o desde VIN a través de un regulador en la placa, o ser suministrado por USB u otro suministro regulado de 5 V.

- **3V3.** Un suministro de 3.3 V generado por el chip FTDI de la placa. La corriente máxima es de 50 mA.

- **GND.** Pines de Tierra.

3.2.4. Memoria

El ATmega168 tiene 16 KB de memoria Flash para almacenar código (de los cuales 2 KB se usa para el «bootloader»). Tiene 1 KB de SRAM y 512 bytes de EEPROM (que puede ser leida y escrita con la librería EEPROM[1]).

3.2.5. Entrada y Salida

Cada uno de los 14 pines digitales del Diecimila puede ser usado como entrada o salida, usando funciones *pinMode()*, *digitalWrite()* y *digitalRead()*[2]. Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna «pull-up» (desconectada por defecto) de 20-50 KOhms. Además, algunos pines tienen funciones especiales:

- **Serial: 0 (Rx) y 1 (Tx).** Usados para recibir (Rx) y transmitir (Tx) datos TTL en serie. Estos pines estan conectados a los pines correspondientes del chip FTDI USB-a-TTL Serie.
- **Interruptores externos: 2 y 3.** Estos pines pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. Mirar la función *attachInterrupt()*[3].
- **PWM: 3, 5, 6, 9, 10 y 11.** Proporcionan salida PWM de 8 bits con la función *analogWrite()*[4].
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Estos pines soportan comunicación SPI, la cual, aunque proporcionada por el hardware subyacente, no está actualmente incluida en el lenguaje Arduino.
- **LED: 13.** Hay un LED empotrado conectado al pin digital 13. Cuando el pin está a valor HIGH, el LED está encendido, cuando el pin está a LOW, está apagado.

El Diecimila tiene 6 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (por ejemplo 1024 valores diferentes). Por defecto miden 5 voltios desde tierra, aunque es posible cambiar el valor más alto de su rango usando el pin ARF y algún código de bajo nivel. Además, algunos pines tienen funcionalidad especializada:

- **I²C: 4 (SDA) y 5 (SCL).** Soportan comunicación I²C (TWI) usando la *libreria Wire*[5].

Hay otro par de pines en la placa:

- **AREF.** Voltaje de referencia para las entradas analógicas. Usado con *analogReference()*[6].
- **Reset.** Pone esta linea a LOW para resetear el microcontrolador. Típicamente usada para añadir un botón de reset a dispositivos que bloquean a la placa principal.

[1] Más información en: `http://www.arduino.cc/en/Reference/EEPROM`
[2] Más información en: `http://www.arduino.cc/en/Reference/`
[3] Más información en: `http://www.arduino.cc/en/Reference`
[4] Más información en: `http://www.arduino.cc/en/Reference`
[5] Más información en: `http://wiring.org.co/reference/libraries/Wire/index.html`
[6] Más información en: `http://www.arduino.cc/en/Reference`

3.2.6. Comunicación

El Arduino Diecimila tiene un numero de infraestructuras para comunicarse con un ordenador, otro Arduino, u otros microcontroladores. El ATmega168 provee comunicación serie UART TTL (5 V), la cual está disponible en los pines digitales 0 (Rx) y 1 (Tx). Un FTDI FT232RL en la placa canaliza esta comunicación serie al USB y los drivers FTDI (incluidos con el software Arduino) proporcionan un puerto de comunicación virtual al software del ordenador. El software Arduino incluye un monitor serie que permite a datos de texto simple ser enviados a y desde la placa Arduino.

Una libreria *SoftwareSerial*[7] permite comunicación serie en cualquiera de los pines digitales del Diecimila.

El ATmega168 también soporta comunicación 12C (TWI) y SPI. El software Arduino incluye una librería Wire para simplificar el uso del bus 12C[8]. Para usar la comunicación SPI, consultar el esquema del ATmega168.

3.2.7. Programación

El Arduino Diecimila puede ser programado con el software Arduino[9].

El ATmega168 del Arduino Diecimila viene con un *bootloader*[10] pregrabado que te permite subirle nuevo código sin usar un programador hardware externo. Se comunica usando el protocolo original STK500.

También puedes saltar el *bootloader* y programar el ATmega168 a través de la cabecera ICSP (In-Circuit Serial Programming)[11].

3.2.8. Reseteo Automático (Software)

En lugar de requerir una pulsación física del botón de reset antes de una subida, el Arduino Diecimila esta diseñado de forma que permite ser reseteado por software en ejecución en una computadora conectada. Una de las lineas de control de flujo de hardware (DTR) del FT232RL esta conectada a la linea de reset del ATmega168 a través de un condensador de 100 nF. Cuando esta linea toma el valor LOW, la linea reset se mantiene el tiempo suficiente para resetear el chip. La version 0009 del software Arduino usa esta capacidad para permitirte cargar código simplemente presionando el botón *upload* en el entorno Arduino. Esto significa que el *bootloader* puede tener un tiempo de espera más corto, mientras la bajada del DTR puede ser coordinada correctamente con el comienzo de la subida.

Esta configuración tiene otras repercusiones. Cuando el Diecimila esta conectado a un ordenador que ejecuta o Mac OS X o Linux, se resetea cada vez que se hace una conexión a él por software (a través de USB). Durante el siguiente medio segundo aproximadamente, el *bootloader* se ejecutará en el Diecimila. Mientras esté programado para ignorar datos «malformados» (por ejemplo, cualquiera excepto una subida de código nuevo), interceptará los primeros bytes de datos enviados a la placa despues de abrir la conexión. Si una rutina que se ejecuta en la placa recibe una configuración una

[7]Más información en: `http://www.arduino.cc/en/Reference/SoftwareSerial`
[8]Para más detalles visitar: `http://wiring.org.co/reference/libraries/Wire/index.html`
[9]Descargar desde: `http://www.arduino.cc/en/Main/Software`
[10]Más información en: `http://www.arduino.cc/en/Tutorial/Bootloader`
[11]Más detalles en:`http://www.arduino.cc/en/Hacking/Programmer`

vez u otros datos cuando empieza, asegurarse de que el software con el que se comunica espera un segundo después de abrir la conexión y antes de enviar estos datos.

3.2.9. Protección de Sobrecarga del USB

El Arduino Diecimila tiene un fusible reseteable que protege tus puertos USB del ordenador de cortes y sobrecargas. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una capa de protección extra. Si más de 500 mA se aplican al puerto USB, el fusible automáticamente romperá la conexión hasta que el corte o la sobrecarga sean eliminados.

3.2.10. Características Físicas

La máxima longitud y anchura del Diecimila PCB son 2.7 y 2.1 pulgadas respectivamente, con el conector USB y el conector de alimentación que se extienden más alla de las primeras dimensiones. Tres agujeros de tornillo permiten a la placa atornillarse a una superficie o caja.

Capítulo 4

SOFTWARE

4.1. Instalar el Software Arduino

Esta sección explica como instalar el software Arduino en un ordenador que ejecute cualquiera de los siguientes Sistemas Operativos: Windows, Mac OS X, GNU/Linux.

Este documento explica como conectar tu placa Arduino al ordenador y cargar tu primera rutina.

Estos son los pasos que seguiremos:

1. Obtener una placa Arduino y un cable.
2. Descargar el entorno Arduino.
3. Instalar los drivers USB.
4. Conectar la placa.
5. Conectar un LED.
6. Ejecutar el entorno Arduino.
7. Subir un programa.
8. Buscar el Led que parpadea.

1.Obtener una placa Arduino y un cable

En este tutorial se asume que estas usando un Arduino Diecimila. Si tienes otra placa, lee la información correspondiente en `http://www.arduino.cc/en/Guide/HomePage`. El Arduino Diecimila es una placa simple que contiene todo lo que necesitas para empezar a trabajar con electrónica y programación de microcontrolador.

También necesitas un cable USB estándar (del tipo que conectarías a una impresora USB, por ejemplo).

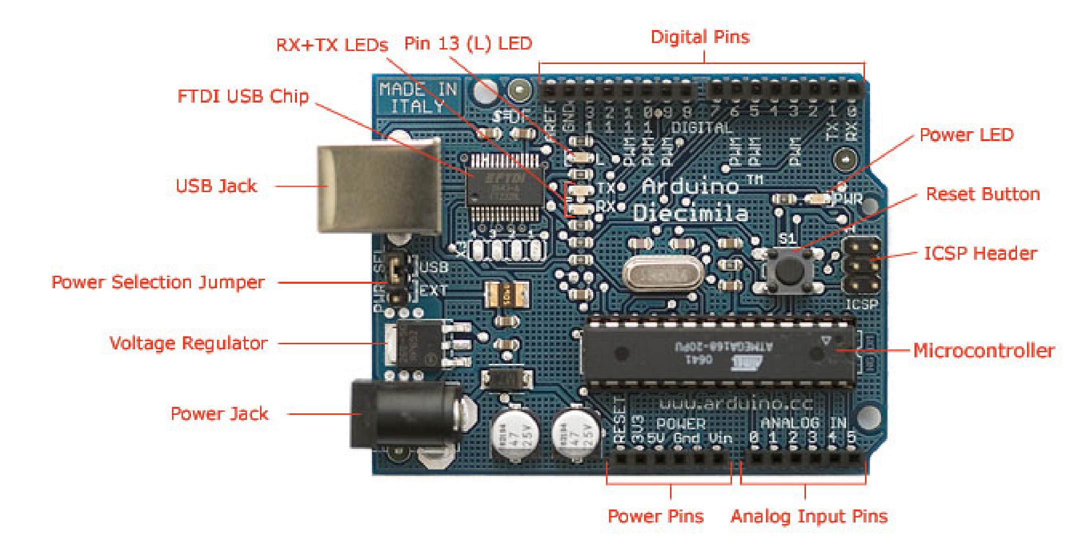


Figura 4.1: Descripción de componentes de la placa Arduino Diecimila

4.1.1. Windows

2.Descargar el entorno Arduino.

Para programar la placa Arduino necesitas el entorno Arduino.

Descarga la última versión desde `http://www.arduino.cc/en/Main/Software`.

Cuando termine la descarga, descomprime el archivo descargado. Asegurate de conservar la estructura de carpetas. Haz doble click en la carpeta para abrirla. Debería haber archivos y subcarpetas en su interior.

3.Instalar los drivers USB.

Si estas usando un Arduino USB, necesitarás instalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio *drivers/FTDI USB Drivers* de la distribución Arduino. En el siguiente paso («Conectar la placa»), se mostrará el asistente para *Añadir Nuevo Hardware de Windows* para estos drivers.

La última versión de los drivers se puede encontrar en `http://www.ftdichip.com/Drivers/VCP.htm`.

4.Conectar la placa.

La fuente de alimentación se selecciona mediante el *jumper* entre los conectores del USB y alimentación. Para alimentar la placa desde el puerto USB (bueno para controlar dispositivos de baja potencia como LEDs), coloca el *jumper* en los dos pines más cercanos al conector USB. Para alimentar la placa desde una fuente externa (6-12 V), coloca el *jumper* en los dos pines más cercanos al conector de alimentación. En cualquier caso, conecta la placa a un puerto USB de tu ordenador.

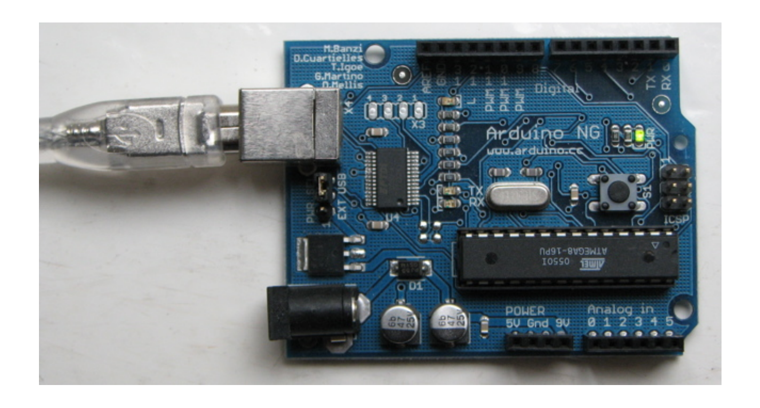


Figura 4.2: Conexión del cable USB a la placa Arduino

El LED de alimentación debería encenderse.

El asistente para *Añadir Nuevo Hardware* debería abrirse. Indícale que no conecte a *Windows Update* y haz click en siguiente.

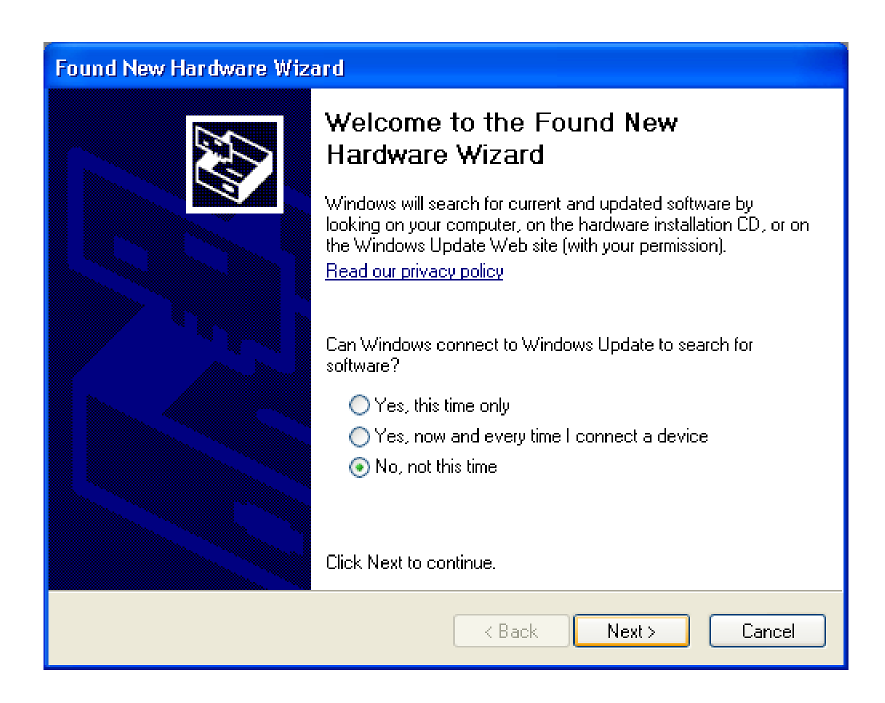


Figura 4.3: Asistente para Nuevo Hardware MS-Windows - Paso 1

Selecciona «Instalar desde una lista o ubicación especificada (Avanzado)» y haz click en siguiente.

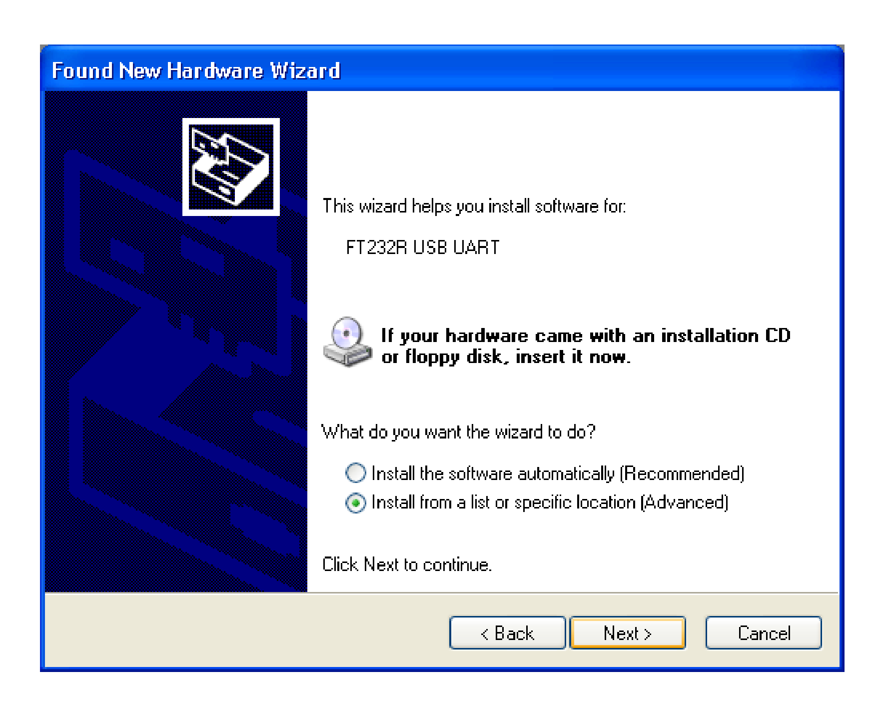


Figura 4.4: Asistente para Nuevo Hardware MS-Windows - Paso 2

Asegurate que «Buscar el mejor driver en estas ubicaciones» está marcado; desmarca «Buscar dispositivos extraibles»; marca «Incluir esta ubicación en la búsqueda» y navega a la ubicación donde descomprimiste los drivers USB en el paso anterior. Haz click en siguiente.

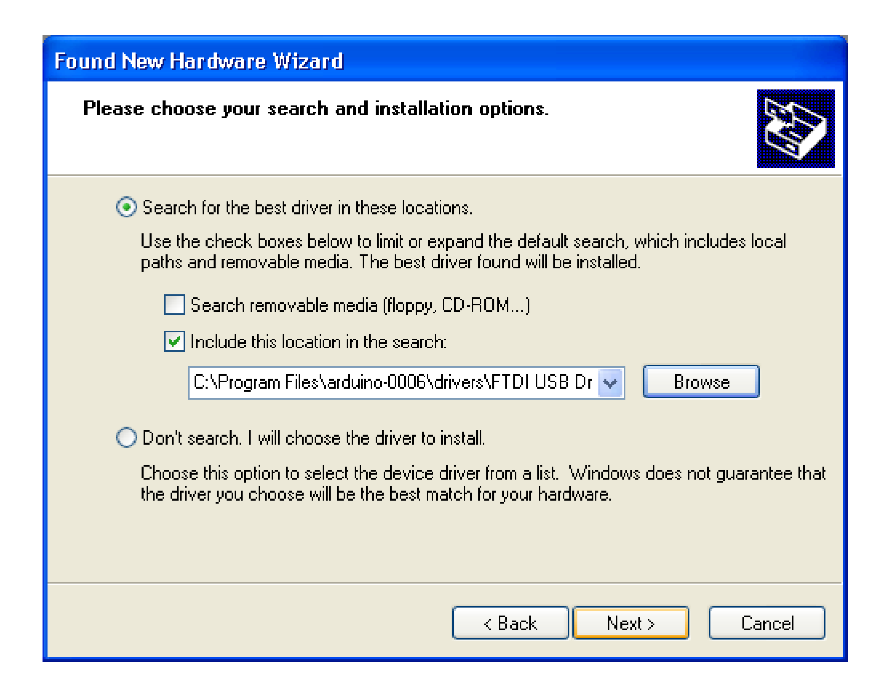


Figura 4.5: Asistente para Nuevo Hardware MS-Windows - Paso 3

El asistente buscará el driver y dirá que se encontró «USB Serial Converter». Haz click en finalizar.

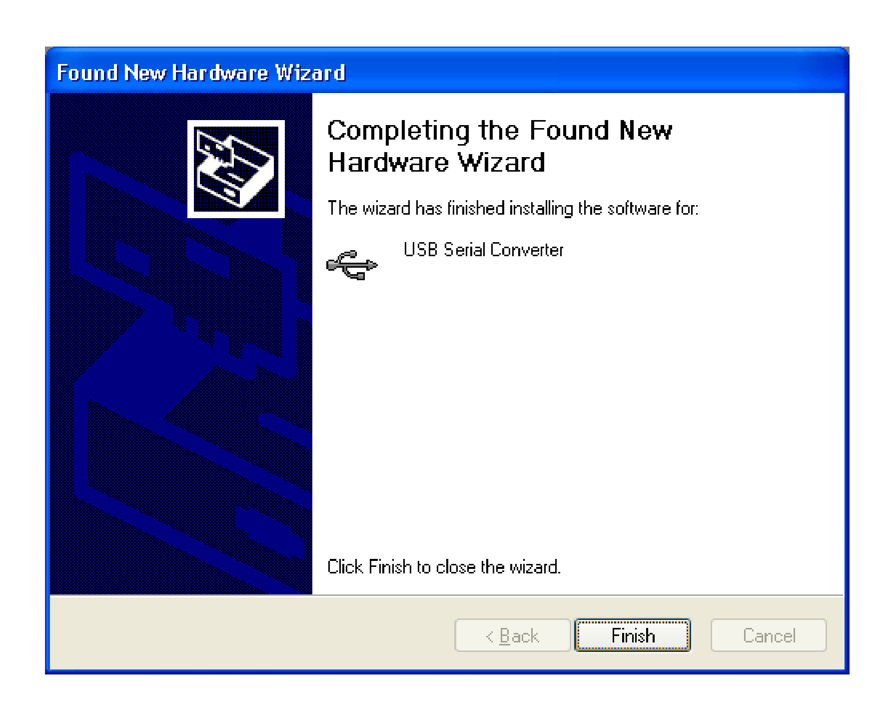


Figura 4.6: Asistente para Nuevo Hardware MS-Windows - Paso 4

El asistente para *Añadir Nuevo Hardware* aparecerá de nuevo. Realiza los mismos pasos. Esta vez, se encontrará un «USB Serial Port».

5. Conectar un LED (si estas usando una placa antigua).

La primera rutina que subirás a la placa Arduino hace parpadear un LED. El Arduino Diecimila (y el Arduino NG original) tiene una resistencia incorporada y un LED en el pin 13. En el Arduino NG Rev. C y placas Arduino pre-NG, sin embargo, el pin 13 no tiene un LED incorporado. En estas placas, necesitarás conectar la patilla positiva (más larga) de un LED al pin 13 y la negativa (más corta) a tierra (marcada como «GND»). Normalmente, también necesitaras usar una resistencia con el LED, pero estas placas tienen una resistencia integrada en el pin 13.

6.Ejecutar el entorno Arduino.

Abrir la carpeta de Arduino y hacer doble click en la aplicación Arduino.

7.Subir un programa.

Abrir la rutina de ejemplo de parpadeo del LED: *File* > *Sketchbook* > *Examples* > *Digital* > *Blink*.

Seleccionar el dispositivo serie de la placa Arduino desde el menu *Herramientas* > *Puerto Serie*. En Windows, este debería ser *COM1* o *COM2* para la placa serie Arduino, o *COM3, COM4* o *COM5* para la placa USB. Para descubrirlo, abrir el *Administrador de Dispositivos de Windows* (En la pestaña *Hardware* o en el *Panel de Control de Sistema).* Buscar un «USB Serial Port» en la sección *Puertos;* esa es la placa Arduino.

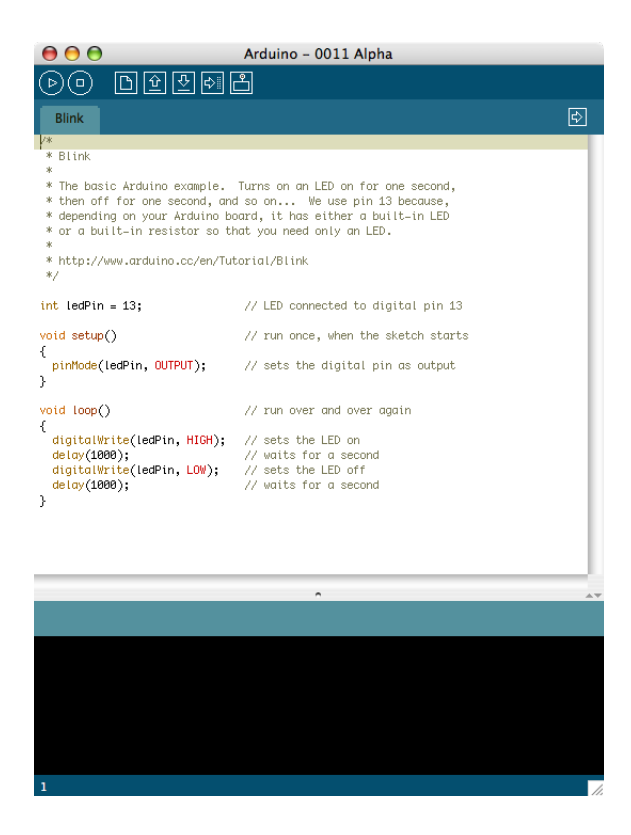


Figura 4.7: Entorno Arduino

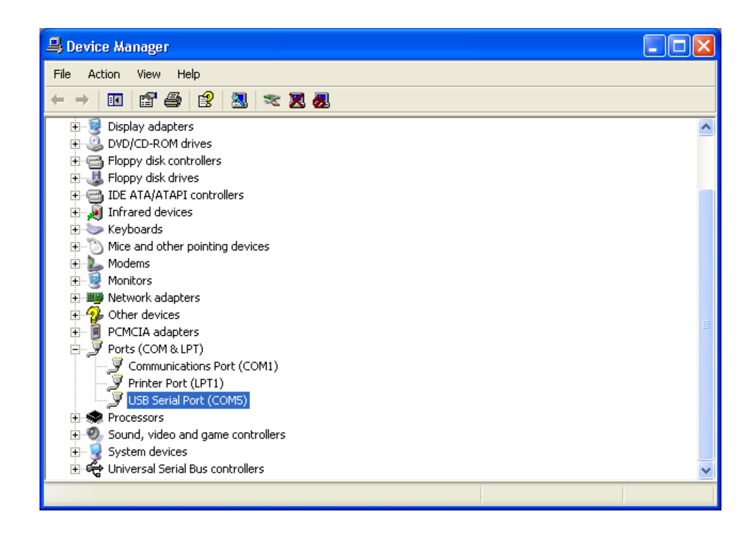


Figura 4.8: Administrador de Dispositivos MS-Windows

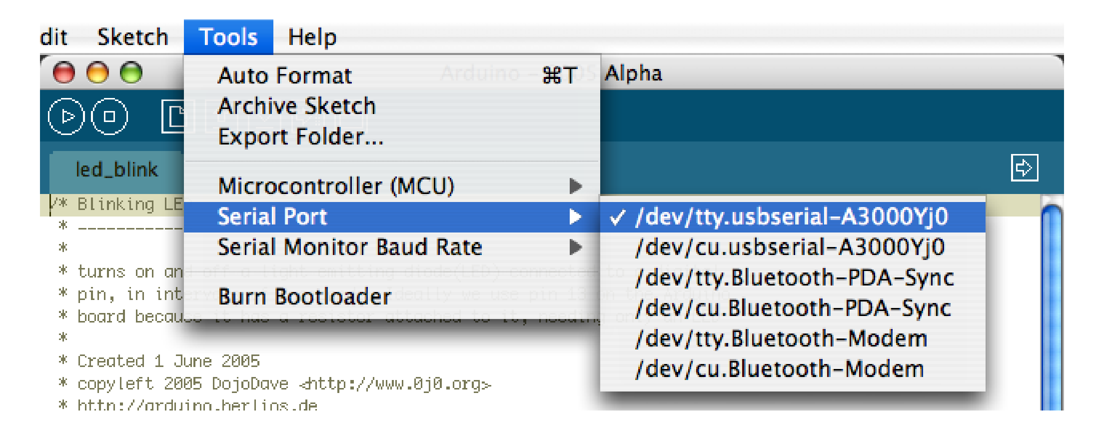


Figura 4.9: Menú de selección de puerto del Entorno Arduino

Asegurarse de que «Arduino Diecimila» está seleccionada en el menú *Tools > Board*.

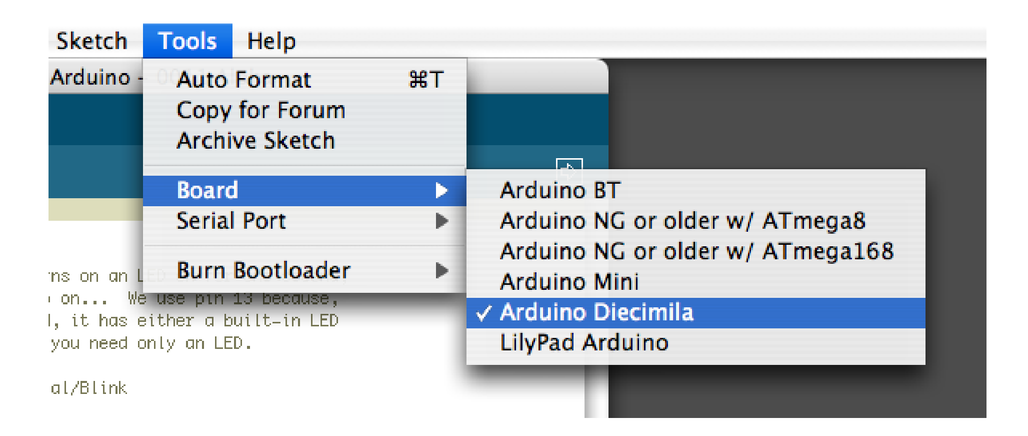


Figura 4.10: Menú de selección de placa del Entorno Arduino

Ahora, simplemente haz click en el botón «Upload» del entorno. Espera unos pocos segundos (deberías ver los LEDs Rx y Tx de la placa iluminándose). Si la carga es correcta, el mensaje «Done uploading» aparacerá en la barra de estado.

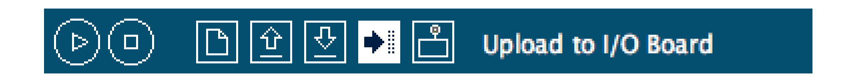


Figura 4.11: Botón de subida de la rutina a la placa

8.Buscar el LED que parpadea.

Unos pocos segundos después de que la subida termine, deberías ver el LED ámbar (amarillo) en la placa empezar a parpadear. Si lo hace ¡enhorabuena! Has conseguido Arduino cargado y ejecutándose.

Si tienes problemas, consulta `http://www.arduino.cc/en/Guide/Troubleshooting`.

4.1.2. MAC OS X (v. 10.3.9 o posterior)

2.Descargar el entorno Arduino.

Para programar la placa Arduino necesitas el entorno Arduino.

Descarga la última versión desde `http://www.arduino.cc/en/Main/Software`.

Cuando termine la descarga, descomprime el archivo descargado. Asegurate de conservar la estructura de carpetas. Haz doble click en la carpeta para abrirla. Debería haber archivos y subcarpetas en su interior.

3.Instalar los drivers USB.

Si estas usando un Arduino USB, necesitarás instalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio *drivers* de la distribución Arduino.

Si tienes un Mac más antiguo como un Powerbook, iBook, G4 o G5, deberías usar los drivers PPC: *FTDIUSBSerialDriver_v2_1_9.dmg*. Si tienes un Mac más nuevo como un MacBook, MacBook Pro o Mac Pro, necesitas los drivers de Intel: *FTDIUSBSerialDriver_v2_2_9_Intel.dmg*. Haz doble click para montar la imagen del disco y ejecutar el *FTDIUSBSerialDriver.pkg* incluido.

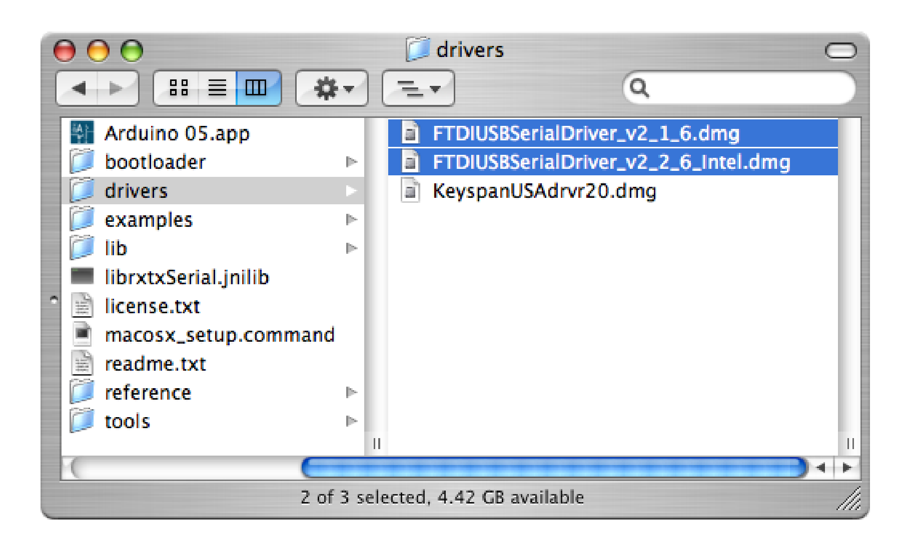


Figura 4.12: Instalación de drivers en Mac OS-X

La última versión de los drivers se puede encontrar en `http://www.ftdichip.com/Drivers/VCP.htm`.

4.Conectar la placa.

La fuente de alimentación se selecciona mediante el *jumper* entre los conectores del USB y alimentación. Para alimentar la placa desde el puerto USB (bueno para controlar dispositivos de baja potencia como LEDs), coloca el *jumper* en los dos pines más cercanos al conector USB. Para alimentar la placa desde una fuente externa (6-12 V), coloca el *jumper* en los dos pines más cercanos al conector de alimentación. En cualquier caso, conecta la placa a un puerto USB de tu ordenador.

El LED de alimentación debería encenderse.

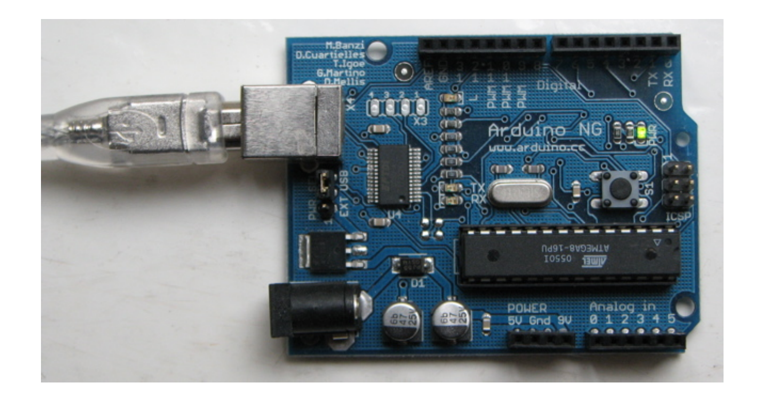


Figura 4.13: Conexión del cable USB a la placa Arduino

5. Conectar un LED (si estas usando una placa antigua).

La primera rutina que subirás a la placa Arduino hace parpadear un LED. El Arduino Diecimila (y el Arduino NG original) tiene una resistencia incorporada y un LED en el pin 13. En el Arduino NG Rev. C y placas Arduino pre-NG, sin embargo, el pin 13 no tiene un LED incorporado. En estas placas, necesitarás conectar la patilla positiva (más larga) de un LED al pin 13 y la negativa (más corta) a tierra (marcada como «GND»). Normalmente, también necesitaras usar una resistencia con el LED, pero estas placas tienen una resistencia integrada en el pin 13.

6.Ejecutar el entorno Arduino.

Abrir la carpeta de Arduino y hacer doble click en la aplicación Arduino.

7.Subir un programa.

Abrir la rutina de ejemplo de parpadeo del LED: *File* > *Sketchbook* > *Examples* > *Digital* > *Blink*.

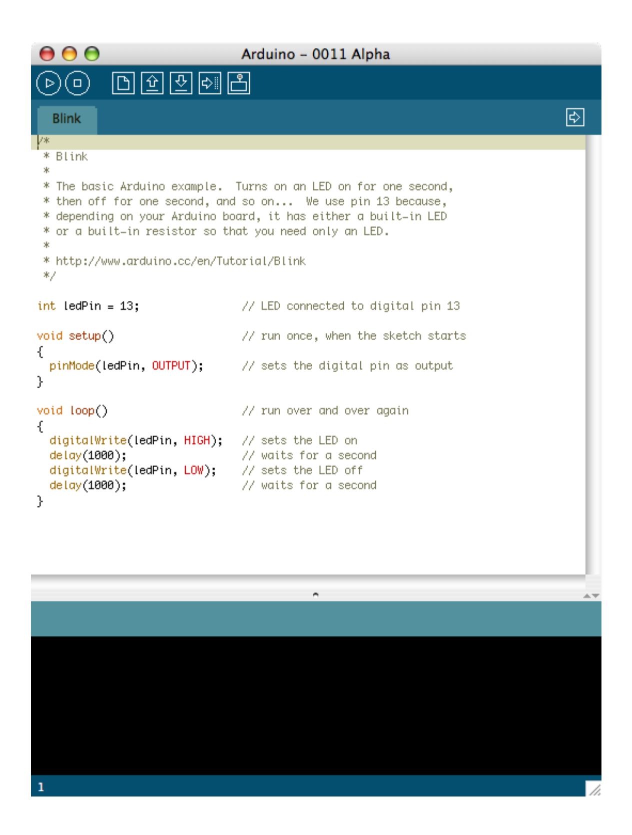


Figura 4.14: Entorno Arduino

Selecciona el dispositivo de la placa Arduino desde el menú *Tools > Serial Port*. En el Mac, debería ser algo con */dev/tty.usbserial*.

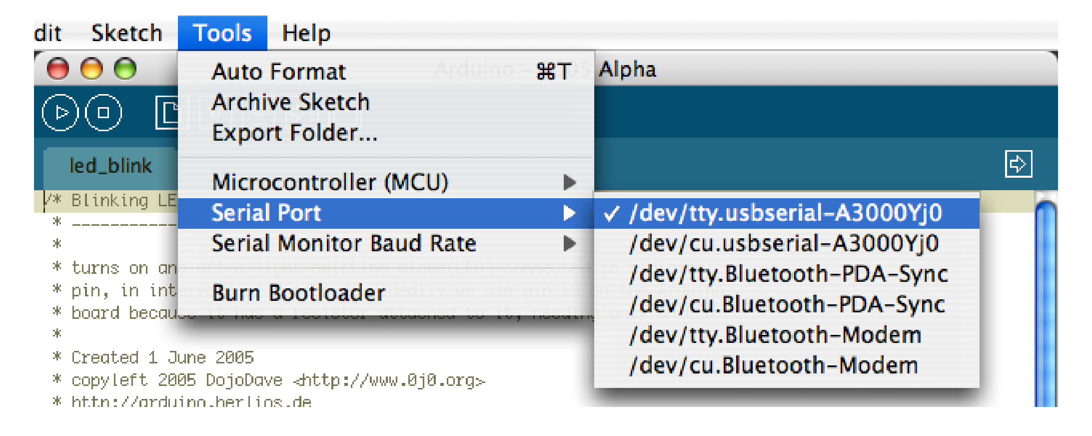


Figura 4.15: Menú de selección de puerto del Entorno Arduino

Asegurate de que «Arduino Diecimila» está seleccionado en el menú *Tools > Board*.

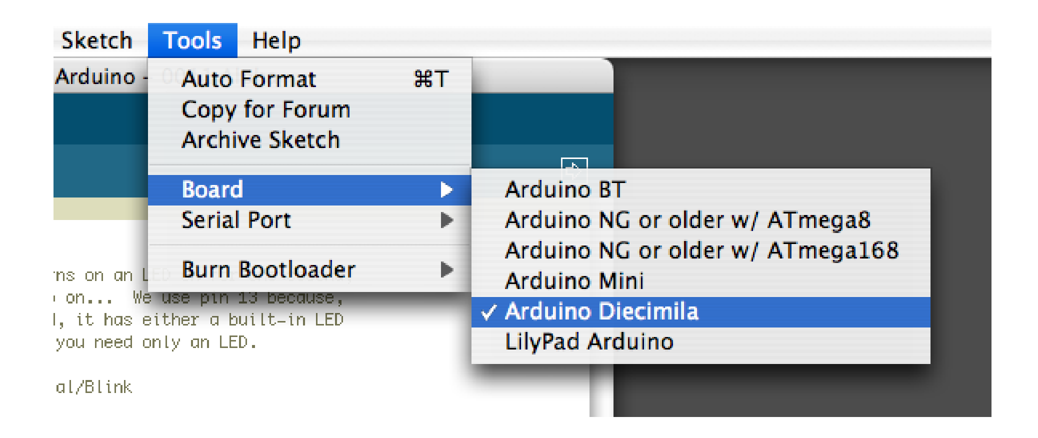


Figura 4.16: Menú de selección de placa del Entorno Arduino

Ahora simplemente haz click en el botón «Upload» en el entorno. Espera unos pocos segundos (deberías ver los LEDs Rx y Tx en la placa iluminándose). Si la subida es correcta, el mensaje «Done uploading» aparecerá en la barra de estado.

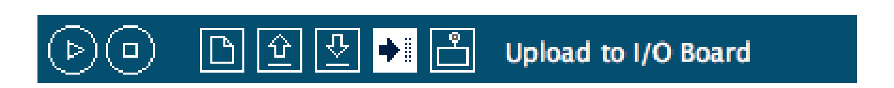


Figura 4.17: Botón de subida de la rutina a la placa

8.Buscar el LED que parpadea.

Unos pocos segundos después de que la subida termine, deberías ver el LED ámbar (amarillo) en la placa empezar a parpadear. Si lo hace ¡enhorabuena! Has conseguido Arduino cargado y ejecutándose.

Si tienes problemas, consulta `http://www.arduino.cc/en/Guide/Troubleshooting`.

4.1.3. GNU/Linux

Estas instrucciones se centran en la distribución Ubuntu[1], para más información sobre cómo instalar el entorno Arduino en otras distribuciones visitar `http://www.arduino.cc/playground/Learning/Linux`.

1. Ejecutar el *Gestor de Paquetes Synaptic* (en *Sistema > Administración*).

2. Primero necesitas habilitar los repositorios «Universe» y «Multiverse» para que puedas acceder a todos los paquetes que necesitas.

    - Ir a *Configuración > Repositorios*.
    - Haz click en *Añadir*.

[1] Más información sobre la distribución en: `http://www.ubuntu.com/`

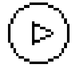
- Marca «Software restringido por copyright o cuestiones legales (multiverse)» y «Software libre mantenido por la comunidad (universe)» y haz click en *Añadir.*
- Haz click en *Cerrar* y click en *Cerrar* en el diálogo «Los repositorios han cambiado».

3. Haz click en el botón *Recargar* de la barra de herramientas.
4. Marca para instalar: «sun-java5-jre», «gcc-avr», «avr-libc».
5. Haz click en *Aplicar* en la barra de herramientas.
6. Haz click en *Aplicar* en el cuadro de diálogo. Esto instalará los paquetes seleccionados.
7. Acepta la licencia de Java.
8. Espera hasta completar la instalación: el cuadro de diálogo dirá «Cambios aplicados». Haz click en *Cerrar.*
9. Cierra *Synaptic.*
10. Descarga la distribición de GNU/Linux de Arduino desde `http://www.arduino.cc/en/Main/Software`. Haz doble click en el archivo *.zip* y arrastra la carpeta que contiene a algún lugar (por ejempl el Escritorio).
11. Ejecuta el *Terminal* (en *Aplicaciones > Accesorios).*
12. Escribe «sudo update-alternatives –config java» y presiona *Enter.* Teclea el número de opción que tiene «java-1.5.0-sun» en él y presiona *Enter.* Esto hará de la versión de Java de Sun la predeterminada de tu sistema (necesaria porque la versión GNU todavía no soporta todo lo necesitado por el entorno Arduino).
13. Haz doble click en «arduino» en el directorio de aplicación de Arduino. Esto debería lanzar un diálogo preguntando dónde guardas los archivos de tus rutinas de Arduino. Un directorio «Arduino» en tu carpeta *home* es la ubicación típica. Haz click en *OK.* El entorno Arduino debería abrirse.

4.2. Introducción al Entorno Arduino

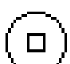
4.2.1. Barra de herramientas

Verify/Compile



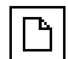
Chequea el código en busca de errores.

Stop



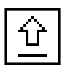
Para el «Serial monitor», o minimiza otros botones.

New



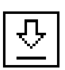
Crea una nueva rutina.

Open



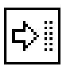
Muestra un menú con todas las rutinas de tu «sketchbook».

Save



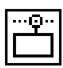
Guarda tus rutinas.

Upload to I/O board



Carga tu código a la placa Arduino I/O. Asegúrate de guardar o verificar tu rutina antes de cargarla.

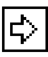
Serial Monitor



Muestra datos serie enviados desde la placa Arduino (placa serie o USB). Para enviar datos a la placa, introduce el texto y haz click en el botón «Send» o presiona «Enter». Elige la velocidad de transmisión de datos desde el desplegable que asigna la velocidad pasada al **Serial.being** en tu rutina. Recuerda que en Mac o GNU/Linux, la placa Arduino se reiniciará (vuelve a ejecutar tu rutina desde del principio) cuando conectes con el «Serial monitor».

Puedes comunicarte también con la placa desde *Processing*, *Flash*, *MaxMSP,* etc (consulta `http://www.arduino.cc/playground/Main/Interfacing` para más detalles).

Tab Menu



Permite gestionar las rutinas con más de un archivo (cada uno de los cuales aparece en su propia pestaña). Estos pueden ser:

- Archivos de código de Arduino (sin extensión).
- Archivos de C (extensión *.c*).
- Archivos de C++ (extensión *.cpp*).
- Archivos de cabecera (extensión *.h*).

4.2.2. Menús

Sketch

- *Verify/Compile:* Comprueba tu rutina para errores.
- *Import Library:* Utiliza una librería en tu rutina. Trabaja añadiendo #**include** en la cima de tu código. Esto añade funcionalidad extra a tu rutina, pero incrementa su tamaño. Para parar de usar una librería, elimina el #**include** apropiado de la cima de tu rutina.
- *Show Sketch Folder:* Abre la carpeta de rutinas en tu escritorio.
- *Add File... :* Añade otro fichero fuente a la rutina. El nuevo archivo aparece en una nueva pestaña en la ventana de la rutina. Esto facilita y agranda proyectos con múltiples archivos fuente. Los archivos pueden ser eliminados de una rutina usando el *Tab Menu.*

Tools

- *Auto Format:* Esto formatea tu código amigablemente.
- *Copy for Discourse:* Copia el código de tu rutina al portapapeles de forma conveniente para postear en un foro, completa con resaltado de sintaxis.
- *Board:* Selecciona la placa que estas usando. Esto controla la forma en que tu rutina es compilada y cargada así como el comportamiento de los elementos del menú *Burn Bootloader.*
- *Serial Port:* Este menú contiene todos los dispositivos serie (reales o virtuales) de tu máquina. Debería actualizarse automáticamente cada vez que abres el nivel superior del menú *Tools.* Antes de subir tu rutina, necesitas seleccionar el elemento de este menú que representa a tu placa Arduino. En el Mac, esto es probablemente algo como **/dev/tty.usbserial-1B1** (para la placa USB), o **/dev/tty.USA19QW1b1P1.1** (para una placa Serie conectada con un adaptador USB-a-Serie Keyspan). En Windows, es probablemente **COM1** o **COM2** (para una placa Serie) o **COM4**, **COM5**, **COM7** o superior (para una placa USB) - para descubrirlo, busca *USB serial device* en la sección puertos del «Gestor de dispositivos de Windows».
- *Burn Bootloader:* Los elementos en este menú te permiten grabar un **bootloader** en tu placa con una variedad de programadores. Esto no es necesario para uso normal de una placa Arduino, pero puede ser útil si encargas ATmegas adicionales o estás construyendo una placa por tu cuenta. Asegurate que has seleccionado la placa correcta del menú *Boards* de antemano. Para grabar un **bootloader** con el AVR ISP, necesitas seleccionar el elemento que corresponde a tu programador del menú *Serial Port.*

4.2.3. Preferencias

Algunas preferencias pueden ser ajustadas en el diálogo *Preferences* (se encuentra bajo el menú *Arduino* en el Mac, o *File* en Windows y GNU/Linux). El resto se puede encontrar en los archivos de preferencias.

Capítulo 5

COMENZANDO CON ARDUINO

5.1. Estructura

La estructura básica del lenguaje de programación Arduino es bastante simple y se organiza en al menos dos partes o funciones que encierran bloques de declaraciones.

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Ambas funciones son requeridas para que el programa funcione.

setup()

La función *setup* debería contener la declaración de cualquier variable al comienzo del programa. Es la primera función a ejecutar en el programa, es ejecutada una vez y es usada para asignar *pinMode* o inicializar las comunicaciones serie.

```
void setup()
{
  pinMode(pin, OUTPUT); //ajusta 'pin' como salida
}
```

loop()

La función *loop* se ejecuta a continuación e incluye el código que se ejecuta contínuamente - leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas Arduino y hace la mayor parte del trabajo.

```
void loop()
{
  digitalWrite(pin, HIGH); //Activa 'pin'
  delay(1000);             //espera un segundo
  digitalWrite(pin, LOW);  //Desactiva 'pin'
  delay(1000);             //espera un segundo
}
```

funciones

Una función es un bloque de código que tiene un nombre y un grupo de declaraciones que se ejecutan cuando se llama a la función. Podemos hacer uso de funciones integradas como *void setup()* y *void loop()* o escribir nuevas.

Las funciones se escriben para ejecutar tareas repetitivas y reducir el desorden en un programa. En primer lugar se declara el tipo de la función, que será el valor retornado por la función (*int, void...*). A continuación del tipo, se declara el nombre de la función y, entre paréntesis, los parámetros que se pasan a la función.

```
type functionName(parameters)
{
  statements;
}
```

La siguiente función *int delayVal()*, asigna un valor de retardo en un programa por lectura del valor de un potenciómetro.

```
int delayVal()
{
  int v;               //crea una variable temporal 'v'
  v = analogRead(pot); //lee el valor del potenciómetro
  v /= 4;              //convierte 0-1023 a 0-255
  return v;            //devuelve el valor final de v
}
```

llaves {}

Las llaves definen el comienzo y el final de bloques de función y bloques de declaraciones como *void loop()* y sentencias *for* e *if*. Las llaves deben estar balanceadas (a una llave de apertura { debe seguirle una llave de cierre }). Las llaves no balanceadas provocan errores de compilación.

```
void loop()
{
  statements;
}
```

El entorno Arduino incluye una práctica característica para chequear el balance de llaves. Sólo selecciona una llave y su compañera lógica aparecerá resaltada.

punto y coma ;

Un punto y coma debe usarse al final de cada declaración y separa los elementos del programa. También se usa para separar los elementos en un bucle *for*.

```
int x = 13; //declara la variable 'x' como el entero 13
```

Nota: Olvidar un punto y coma al final de una declaración producirá un error de compilación.

bloques de comentarios /*...*/

Los bloques de comentarios, o comentarios multilínea, son areas de texto ignoradas por el programa y se usan para grandes descripciones de código o comentarios que ayudan a otras personas a entender partes del programa. Empiezan con /* y terminan con */ y pueden abarcar multiples lineas.

```
/*
 este es un bloque de comentario encerrado
 no olvides cerrar el comentario
 tienen que estar balanceados!
*/
```

Como los comentarios son ignorados por el programa y no ocupan espacio en memoria deben usarse generósamente y también pueden usarse para «comentar» bloques de código con propositos de depuración.

comentarios de línea //

Comentarios de una línea empiezan con // y terminan con la siguiente línea de código. Como el bloque de comentarios, son ignorados por el programa y no toman espacio en memoria.

```
// este es un comentario de una línea
```

Comentarios de una línea se usan a menudo despues de declaraciones válidas para proporcionar más información sobre qué lleva la declaración o proporcionar un recordatorio en el futuro.

5.2. Variables

Una variable es una forma de llamar y almacenar un valor numérico para usarse después por el programa. Como su nombre indica, las variables son números que pueden cambiarse continuamente al contrario que las constantes, cuyo valor nunca cambia. Una variable necesita ser declarada y, opcionalmente, asignada al valor que necesita para ser almacenada.

```
int inputVariable = 0;         //declara una variable y asigna el valor a 0
inputVariable = analogRead(2); //ajusta la variable al valor del pin
                               //analógico 2
```

Una vez que una variable ha sido asignada, o reasignada, puedes testear su valor para ver si cumple ciertas condiciones, o puedes usarlo directamente.

```
if(inputVariable < 100)  //comprueba si la variable es menor que 100
{
  inputVariable = 100;   //si es cierto asigna el valor 100
}
delay(inputVariable);    //usa la variable como retardo
```

declaración de variable

Todas las variables tienen que ser declaradas antes de que puedan ser usadas. Declarar una variable significa definir su tipo de valor, como *int*, *long*, *float*, etc., definir un nombre específico, y, opcionalmente, asignar un valor inicial. Esto sólo necesita hacerse una vez en un programa pero el valor puede cambiarse en cualquier momento usando aritmética y varias asignaciones.

```
int inputVariable = 0;
```

Una variable puede ser declarada en un número de posiciones en todo el programa y donde esta deficinición tiene lugar determina que partes del programa pueden usar la variable.

ámbito de la variable

Una variable puede ser declarada al comienzo del programa antes del *void setup()*, localmente dentro de funciones, y algunas veces en un bloque de declaración, por ejemplo bucles *for*. Donde la variable es declarada determina el ámbito de la variable, o la abilidad de ciertas partes de un programa de hacer uso de la variable.

Una variable global es una que puede ser vista y usada por cualquier función y declaración en un programa. Esta variable se declara al comienzo del programa, antes de la función *setup()*.

Una variable local es una que se define dentro de una función o como parte de un bucle *for*. Sólo es visible y sólo puede ser usada dentro de la función en la cual fue declarada. Además, es posible tener dos o más variables del mismo nombre en diferentes partes del programa que contienen diferentes valores.

```
int value; //'value' es visible por cualquier función

void setup()
{
  //no se necesita setup
}

void loop()
{
  for(int i=0; i<20;) //'i' es sólo visible dentro del bucle for
  {
    i++;
```

```
  }
  float f; //'f' es sólo visible dentro de loop
}
```

5.3. Tipos de datos

byte

Byte almacena un valor numérico de 8 bits sin puntos decimales. Tienen un rango de 0 a 255.

```
byte someVariable = 180; //declara 'someVariable' como un tipo byte
```

int

Enteros son los tipos de datos primarios para almacenamiento de números sin puntos decimales y almacenan un valor de 16 bits con un rango de -32,768 a 32,767.

```
int someVariable = 1500; //declara 'someVariable' como tipo int
```

long

Tipo de datos de tamaño extendido para enteros largos, sin puntos decimales, almacenados en un valor de 32 bits con un rango de -2,146,483,648 a 2,147,483,647.

```
long someVariable = 90000; //declara 'someVariable' como tipo long
```

float

Un tipo de datos para números en punto flotante, o números que tienen un punto decimal. Los números en punto flotante tienen mayor resolución que los enteros y se almacenan como valor de 32 bits con un rango de -3.4028235E+38 a 3.4028235E+38.

```
float someVariable = 3.14; //declara 'someVariable' como tipo float
```

arrays

Un array es una colección de valores que son accedidos con un índice numérico. Cualquier valor en el array debe llamarse escribiendo el nombre del array y el índice numérico del valor. Los arrays están indexados a cero, con el primer valor en el array comenzando con el índice número 0. Un array necesita ser declarado y opcionalmente asignarle valores antes de que puedan ser usados.

```
int myArray[] = {value0, value1, value2...};
```

Asimismo es posible declarar un array declarando el tipo del array y el tamaño y luego asignarle valores a una posición del índice.

```
int myArray[5];  //declara un array de enteros con 6 posiciones
myArray[3] = 10; //asigna a la cuarta posición del índice el valor 10
```

Para recibir un valor desde un array, asignamos una variable al array y la posición del índice:

```
x = myArray[3]; //x ahora es igual a 10
```

5.4. Aritmética

Los operadores aritméticos incluyen suma, resta, multiplicación y división. Retornan la suma, diferencia, producto o cociente (respectivamente) de dos operandos.

```
y = y+3;
x = x-7;
i = j*6;
r = r/5;
```

La operación es llevada a cabo usando del tipo de datos de los operandos, así 9/4 devuelve 2 en lugar de 2.25. Si los operandos son de tipos diferentes, el tipo mayor es usado para el cálculo.

Nota: Usar el operador *cast*, por ejemplo *(int)myFloat* para convertir un tipo de variable a otro al vuelo.

asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una asignación de variable. Estas son muy frecuentemente encontradas en bucles *for*. Las asignaciones compuestas más comunes incluyen:

```
x++;    //lo mismo que x = x+1
x--;    //lo mismo que x = x-1
x += y; //lo mismo que x = x+y
x -= y; //lo mismo que x = x-y
x *= y; //lo mismo que x = x*y
x /= y; //lo mismo que x = x/y
```

operadores de comparación

Las comparaciones de una variable o constante con otra se usan a menudo en declaraciones *if* para comprobar si un condición específica es cierta.

```
x == y; //x es igual a y
x != y; //x no es igual a y
x < y;  //x es menor que y
x > y;  //x es mayor que y
x <= y; //x es menor o igual que y
x >= y; //x es mayor o igual que y
```

operadores lógicos

Los operadores lógicos son normalmente una forma de comparar dos expresiones y devuelven TRUE o FALSE dependiendo del operador. Hay tres operadores lógicos, AND, OR y NOT, que se usan a menudo en declaraciones *if*.

```
//AND logico:
if(x>0 && x<5)  //verdadero sólo si las dos expresiones son ciertas

//OR logico:
if(x>0 || y>0)  //verdadero si al menos una expresion es cierta

//NOT logico:
if(!(x>0))      //verdadero sólo si la expresión es falsa
```

5.5. Constantes

El lenguaje Arduino tiene unos cuantos valores predefinidos que se llaman constantes. Se usan para hacer los programas más legibles. Las constantes se clasifican en grupos.

true/false

Estas son constantes Booleanas que definen niveles lógicos. FALSE se define como 0 (cero) mientras TRUE es 1 o un valor distinto de 0.

```
if(b == TRUE)
{
  doSomething;
}
```

high/low

Estas constantes definen los niveles de pin como HIGH o LOW y se usan cuando se leen o se escriben los pines digitales. HIGH esta definido como el nivel 1 lógico, ON ó 5 V, mientras que LOW es el nivel lógico 0, OFF ó 0 V.

```
digitalWrite(13, HIGH);
```

input/output

Constantes usadas con la función *pinMode()* para definir el modo de un pin digital como INPUT u OUTPUT.

```
pinMode(13, OUTPUT);
```

5.6. Control de flujo

if

Las sentencias *if* comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa ignora la sentencia.

```
if(someVariable ?? value)
{
  doSomething;
}
```

Nota: Cuídate de usar «=» en lugar de «==» dentro de la declaración de la sentencia *if*.

if... else

if... else permite tomar decisiones «este - o este».

```
if(inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

else puede preceder a otra comprobación *if*, por lo que multiples y mutuas comprobaciones exclusivas pueden ejecutarse al mismo tiempo.

```
if(inputPin < 500)
{
  doThingA;
}
else if(inputPin >= 1000)
{
  doThingB;
}
else
{
  doThingC;
}
```

for

La sentencia *for* se usa para repetir un bloque de declaraciones encerradas en llaves un número específico de veces. Un contador de incremento se usa a menudo para incrementar y terminar el bucle. Hay tres partes seperadas por punto y coma (;), en la cabecera del bucle.

```
for(inicializacion; condicion; expresion)
{
  doSomething;
}
```

La inicialización de una variable local, o contador de incremento, sucede primero y una sola una vez. Cada vez que pasa el bucle, la condición siguiente es comprobada. Si la condición devuelve TRUE, las declaraciones y expresiones que siguen se ejecutan y la condición se comprueba de nuevo. Cuando la condición se vuelve FALSE, el bucle termina.

```
for(int i=0; i<20; i++)      //declara i, comprueba si es menor
{                            //que 20, incrementa i en 1
  digitalWrite(13, HIGH);    //activa el pin 13
  delay(250);                //pausa por un 1/4 de segundo
  digitalWrite(13, LOW);     //desactiva el pin 13
  delay(250);                //pausa por un 1/4 de segundo
}
```

while

El bucle *while* se repetirá continuamente, e infinitamente, hasta que la expresión dentro del paréntesis se vuelva falsa. Algo debe cambiar la variable testeada, o el bucle *while* nunca saldrá. Esto podría estar en tu código, como por ejemplo una variable incrementada, o una condición externa, como un sensor de comprobación.

```
while(someVariable ?? value)
{
  doSomething;
}

while(someVariable < 200)      //comprueba si es menor que 200
{
  doSomething;                 //ejecuta las sentencias encerradas
  someVariable++;              //incrementa la variable en 1
}
```

do. . . while

El bucle *do. . . while* es un bucle que trabaja de la misma forma que el bucle *while*, con la excepción de que la condición es testeada al final del bucle, por lo que el bucle *do. . . while* siempre se ejecutará al menos una vez.

```
do
{
  doSomething;
}while(someVariable ?? value);

do
{
  x = readSensors();      //asigna el valor de readSensors() a x
  delay(50);              //pausa de 50 milisegundos
}while(x < 100);          //repite si x es menor que 100
```

5.7. E/S digital

pinMode(pin, mode)

Se usa en *void setup()* para configurar un pin específico para que se comporte o como INPUT o como OUTPUT.

```
pintMode(pin, OUTPUT);  //ajusta 'pin' como salida
```

Los pines digitales de Arduino estan ajustados a INPUT por defecto, por lo que no necesitan ser declarados explícitamente como entradas con *pinMode()*. Los pines configurados como INPUT se dice que están e un estado de alta impedancia.

Hay también convenientes resistencias de *pull-up* de 20KOhm, integradas en el chip ATmega que pueden ser accedidas por software. A estas resistencias *pull-up* integradas se accede de la siguiente manera.

```
pinMode(pin, INPUT);       //ajusta 'pin' como entrada
digitalWrite(pin, HIGH);   //activa la resistencia de pull-up
```

Las resistencias de *pull-up* se usarían normalmente para conectar entradas como interruptores.

Los pines configurados como OUTPUT se dice que están en un estado de baja impedancia y pueden proporcionar 40 mA a otros dispositivos/circuitos.

Nota: Cortocircuitos en los pines de Arduino o corriente excesiva puden dañar o destruir el pin de salida, o dañar el chip ATmega. A menudo es una buena idea conectar un pin OUTPUT a un dispositivo externo en serie con una resistencia de 470Ohm o 1KOhm.

digitalRead(pin)

Lee el valor desde un pin digital especificado con el resultado HIGH o LOW. El pin puede ser especificado o como una variable o como una constante (0 - 13).

```
value = digitalRead(Pin);   //ajusta 'value' igual al pin de entrada
```

digitalWrite(pin, value)

Devuelve o el nivel lógico HIGH o LOW a (activa o desactiva) un pin digital especificado. El pin puede ser especificado como una variable o constante (0 - 13).

```
digitalWrite(pin, HIGH);   //ajusta 'pin' a HIGH

//Ejemplo de programa
int led = 13;   //conecta 'led' al pin 13
int pin = 7;    //conecta 'pushbutton' al pin 7
int value = 0;  //variable para almacenar el valor leido

void setup()
{
  pinMode(led, OUTPUT);   //ajusta el pin 13 como salida
  pinMode(pin, INPUT);    //ajusta el pin 7 como entrada
}

void loop()
{
  value = digitalRead(pin);   //ajusta 'value' igual al pin de entrada
  digitalWrite(led, value);   //ajusta 'led' al valor del boton
}
```

5.8. E/S analógica

analogRead(pin)

Lee el valor desde un pin analógico especificado con una resolución de 10 bits. Esta función sólo trabaja en los pines analógicos (0 - 5). Los valores enteros devueltos están en el rango de 0 a 1023.

```
value = analogRead(pin);   //ajusta 'value' igual a 'pin'
```

Nota: Los pines analógicos al contrario que los digitales, no necesitan ser declarados al principio como INPUT u OUTPUT.

analogWrite(pin, value)

Escribe un valor pseudo analógico usando *modulación por ancho de pulso* («PWM» en inglés) a un pin de salida marcado como PWM. En los Arduinos más nuevos con el chip ATmega168, esta función trabaja en los pines 3, 5, 6, 9, 10 y 11. Los Arduinos más antiguos con un ATmega8 sólo soporta los pines 9, 10 y 11. El valor puede ser especificado como una variable o constante con un valor de 0 a 255.

```
analogWrite(pin, value);   //escribe 'value' al 'pin' analogico
```

Valor	Nivel de salida
0	0 V (t)
64	0 V (3/4 de t) y 5 V (1/4 de t)
128	0 V (1/2 de t) y 5 V (1/2 de t)
192	0 V (1/4 de t) y 5 v (3/4 de t)
255	5 V (t)

Cuadro 5.1: Relación valor-salida con *analogWrite()*

El valor de salida varía de 0 a 5 V según el valor de entrada (de 0 a 255) en función del tiempo de pulso. Si t es el tiempo de pulso, la tabla 5.1 muestra la equivalencia entre el valor y la salida en función del tiempo.

Como esta es una función hardware, el pin generará una onda estática después de una llamada a *analogWrite* en segundo plano hasta la siguiente llamada a *analogWrite* (o una llamada a *digitalRead* o *digitalWrite* en el mismo pin).

```
int led = 10;               //LED con una resistencia de 220ohm en el pin 10
int pin = 0;                //potenciometro en el pin analogico 0
int value;                  //valor para lectura

void setup(){}              //setup no es necesario

void loop()
{
  value = analogRead(pin);  //ajusta 'value' igual a 'pin'
  value /= 4;               //convierte 0-1023 a 0-255
  analogWrite(led, value);  //saca la señal PWM al led
}
```

5.9. Tiempo

delay(ms)

Pausa tu programa por la cantidad de tiempo especificada en milisegundos, donde 1000 es igual a 1 segundo.

```
delay(1000);   //espera por un segundo
```

millis()

Devuelve el número de milisegundos desde que la placa Arduino empezó a ejecutar el programa actual como un valor *long* sin signo.

```
value = millis(); //ajusta 'value' igual a millis()
```

Nota: Este número se desbordará (resetear de nuevo a cero), después de aproximadamente 9 horas.

5.10. Matemáticas

min(x,y)

Calcula el mínimo de dos números de cualquier tipo de datos y devuelve el número más pequeño.

```
value = min(value, 100);   //asigna a 'value' al más pequeño de 'value' o 100,
                           //asegurandose que nunca superara 100.
```

max(x,y)

Calcula el máximo de dos números de cualquier tipo de datos y devuelve el número más grande.

```
value = max(value, 100);    //asigna a 'value' al más grande de 'value' o 100,
                            //asegurandose que es al menos 100.
```

5.11. Aleatorio

randomSeed(seed)

Asigna un valor, o semilla («seed» en inglés), como el punto de partida para la función *random()*.

```
randomSeed(value);   //asigna 'value' como la semilla aleatoria
```

Como el Arduino es incapaz de crear un número verdaderamente aleatorio, *randomSeed* te permite colocar una variable, constante, u otra función dentro de la función *random*, lo cual ayuda a generar mas números *«random»* aleatorios. Existen una variedad de diferentes semillas, o funciones, que pueden ser usadas en esta función incluyendo *millis()* o incluso *analogRead()* para leer ruido eléctrico a través de un pin analógico.

random(max)
random(min, max)

La función *random* te permite devolver números pseudo aleatorios en un rango especificado por los valores *min* y *max*.

```
value = random(100, 200);   //asigna a 'value' un número aleatorio
                            //entre 100 y 200.
```

Nota: Usar esto después de usar la función *randomSeed()*.

```
int randNumber;                   //variable para almacenar el valor
                                  //aleatorio
int led = 10;                     //LED con una resistencia de 220ohm
                                  //en el pin 10
```

```
void setup(){}                    //setup no es necesario

void loop()
{
  randomSeed(millis());           //asigna millis() como semilla
  randNumber = random(255);       //numero aleatorio de 0 a 255
  analogWrite(led, randNumber);   //salida de la señal PWM
  delay(500);
}
```

5.12. Serie

Serial.begin(rate)

Abre el puerto serie y asigna la tasa de baudios para la transmisión de datos serie. La típica tasa de baudios para comunicarse con el ordenador es 9600 aunque otras velocidades están soportadas.

```
void setup()
{
  Serial.begin(9600);  //abre el puerto serie
                       //ajusta la tasa de datos a 9600 bps
}
```

Nota: Cuando se usa la comunicación serie, los pines digitales 0 (Rx) y 1 (Tx) no pueden ser usados al mismo tiempo.

Serial.println(data)

Imprime datos al puerto serie, seguido de un retorno de carro y avance de línea automáticos. Este comando toma la misma forma que *Serial.print()*, pero es más fácil para leer datos en el *Serial Monitor*[1].

```
Serial.println(analogValue);   //envia el valor de 'analogValue'

//Ejemplo de aplicacion
void setup()
{
  Serial.begin(9600);              //ajusta al serie a 9600 bps
}

void loop()
{
  Serial.println(analogRead(0));  //envia valor analogico
  delay(1000);                     //pausa por 1 segundo
}
```

[1]Más información en: *4.2 Introducción al Entorno Arduino*

Bibliografía

[1] EVANS, Brian W., (2007) *Arduino Programming Notebook.*

[2] SANTO ORCERO, David, (2007), «Hardware Libre», *Todo Linux*, Madrid: Studio Press. Pp: 12-17.

[3] ARDUINO - Wikipedia, the free encyclopedia, (última modificación, Marzo 2008). Disponible en: `http://en.wikipedia.org/wiki/Arduino`

[4] ARDUINO : Homepage, (última modificación, Marzo 2006). Disponible en: `http://www.arduino.cc/es/`

[5] ARDUINO - Homepage, (última modificación, Julio 2008). Disponible en: `http://www.arduino.cc`

[6] WIRING, (última modificación, Junio 2008). Disponible en: `http://www.wiring.org.co`

Apéndice A

Ejemplos de Aplicación con Arduino

A.1. Salida digital

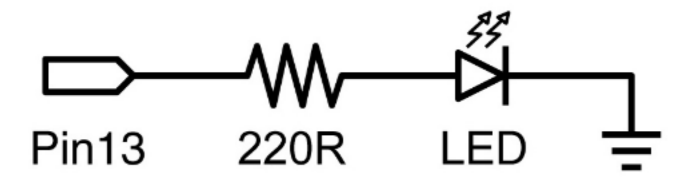


Figura A.1: Esquema de salida digital

Este es el programa básico «hola mundo» usado simplemente para activar o desactivar algo. En este ejemplo, un LED está conectado al pin 13, y parpadea cada segundo. La resistencia puede omitirse en este pin ya que el Arduino tiene una integrada.

```
int ledPin = 13;              //LED en el pin digital 13

void setup()                  //ejecutar una vez
{
  pinMode(ledPin, OUTPUT);    //asigna al pin 13 como salida
}

void loop()                   //ejecutar una y otra vez
{
  digitalWrite(ledPin, HIGH); //activa el LED
  delay(1000);                //pausa 1 segundo
  digitalWrite(ledPin, LOW);  //desactiva el LED
  delay(1000);                //pausa 1 segundo
}
```

A.2. Entrada digital

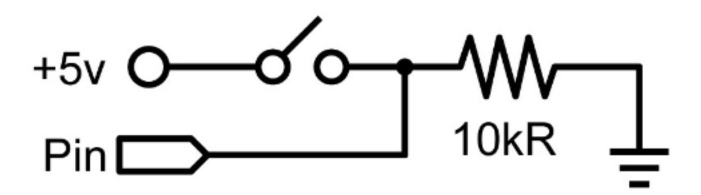


Figura A.2: Esquema de entrada digital

Esta es la forma más simple de entrada con sólo dos estados posibles: ON u OFF. Este ejemplo lee un interruptor simple o pulsador conectado al pin 2. Cuando el interruptor está cerrado el pin de entrada leerá HIGH y activará un LED.

```
int ledPin = 13;                   //pin de salida para el LED
int inPin = 2;                     //pin de entrada (para un interruptor)

void setup()
{
  pinMode(ledPin, OUTPUT);         //declara LED como salida
  pinMode(inPin, INPUT);           //declara el interruptor como entrada
}

void loop()
{
  if(digitalRead(inPin == HIGH)    //comprueba si la entrada esta a HIGH
  {
    digitalWrite(ledPin, HIGH);    //activa el LED
    delay(1000);                   //pausa 1 segundo
    digitalWrite(ledPin, LOW);     //desactiva el LED
    delay(1000);                   //pausa 1 segundo
  }
}
```

A.3. Salida PWM



Figura A.3: Esquema de salida PWM

La *modulación de ancho de pulso* (PWM) es una forma de «falsificar» una salida analógica por la salida pulsante. Esto podría usarse para atenuar e iluminar un LED o posteriormente controlar un servomotor. El siguiente ejemplo ilumina y atenua lentamente un LED usando bucles *for*.

```
int ledPin = 9;              //pin PWM para el LED

void setup(){}

void loop()
{
  for(int i=0; i<=255; i++)   //incrementa el valor para i
  {
    analogWrite(ledPin, i);   //asigna el nivel de brillo a i
    delay(100);               //pausa 100 ms
  }
  for(int i=255; i>=0; i--)   //decrementa el valor para i
  {
    analogWrite(ledPin, i);   //asigna el nivel de brillo a i
    delay(100);               //pausa 100 ms
  }
}
```

A.4. Entrada de potenciometro

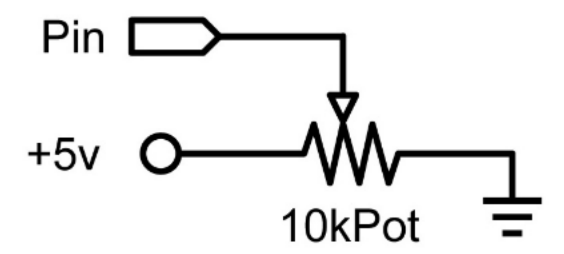


Figura A.4: Esquema de entrada de potenciometro

Usando un potenciometro y una de los pines de conversión analógico-digital (ADC) de Arduino es posible leer valores de 0 a 1024. El siguiente ejemplo usa un potenciometro para controlar una frecuencia de parpadeo de un LED.

```
int potPin = 0;             //pin de entrada para el potenciometro
int ledPin = 13;            //pin de salida para el LED

void setup()
{
  pinMode(ledPin, OUTPUT);  //declara ledPin como OUTPUT
```

```
}

void loop()
{
  digitalWrite(ledPin, HIGH);  //activa ledPin
  delay(analogRead(potPin));   //pausa el programa
  digitalWrite(ledPin, LOW);   //desactiva ledPin
  delay(analogRead(potPin));   //pausa el programa
}
```

Apéndice B

Esquemático de Arduino Diecimila

